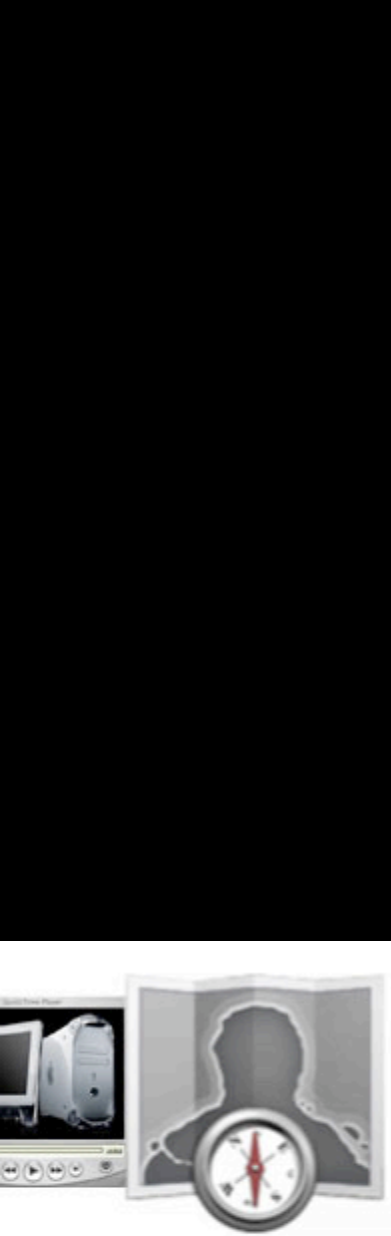


Cocoa Controls and Cocoa Accessibility

Session 304

Cocoa Controls and Cocoa Accessibility

Chuck Pisula
Cocoa Frameworks Engineer

Introduction

- Cocoa provides a rich set of controls
- Most of your app built with existing controls
- Controls are easy to use and customize
- Writing your own control is also easy

What You'll Learn

- Control concepts and API
- How to use and extend existing controls
- Issues involved in writing your own control
- Accessibility considerations

Focus: NSControl and NSCell

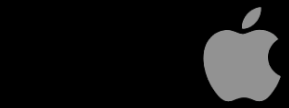
- Many UI elements are NSControls
  - Buttons, text fields, sliders
  - Table view, browser
- Some UI elements are not NSControls
  - Open panel, Color panel
  - Movie view, Toolbar
- Today we are focusing on NSControl

Control-Cell Symbiosis

- NSControls and NSCells work together
- NSControl inherits from NSView
  - Can have one or many NSCells
  - Most tasks are handled by the NSCell
- NSCell inherits from NSObject
  - Handles drawing, mouse tracking, etc . . .
  - Location is provided by its NSControl

Why NSCell?

- Separating drawing and events has advantages
- NSCells are lightweight objects
- Easy to create a group of multiple cells
    - NSTableView, NSMatrix manage groups of multiple dissimilar cells



NSControl

- NSControl manages one or many cells
- Provides location in view hierarchy for each
- Division of labor
  - NSCells implement drawing, event handling, set methods . . .
  - NSControls provide functionality that is cell independent
    - E.g., row selection in a table

NSCell

- Handle drawing and events for their control

```
-(void)drawWithFrame:(NSRect)cellFrame
                inView:(NSView *)controlView;
```

- Handle many other things for their control
  - Setting values, etc . . .
- Usually have many customizable attributes

Similar API

- NSControl methods are often just covers
  - Handled by its cell, or selected cell
- Common API
  - Setting/getting value
  - Target/Action
  - Control size property
  - Context menus
  - Text editing

Values

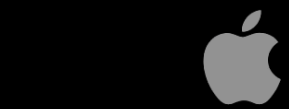
- Setting Values

```
-(void)setFloatValue:(float)f;
-(void)setStringValue:(NSString *)s;
-(void)setAttributedStringValue:(NSAttributedString *)s;
```

- Getting Values

```
-(float)floatValue;
-(NSString *)stringValue;
-(NSAttributedString *)attributedStringValue;
```

- Controls implement each method

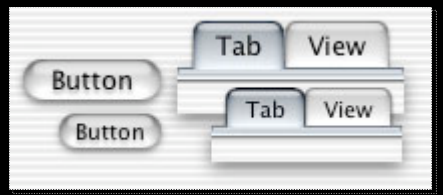


Control Size

- Usually implemented by the NSCell
- Most cells come in two standard sizes

```
-(void)setControlSize:(NSControlSize)controlSize;

typedef {
    NSRegularControlSize,
    NSSmallControlSize
} NSControlSize;
```

- Examples: NSButton, NSTabView



Target/Action

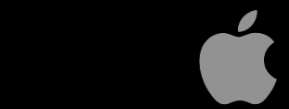
- Cells send action in response to user actions

```
// Target / Action for a matrix of sliders
-(void)sliderValueChanged:(id)sender {
    NSSliderCell *sliderCell = [sender selectedCell];
    NSLog(@"value = %.2f", [sliderCell floatValue]);
}
```

- 'sender' argument is always the NSControl
- Some NSControls have own action/double action

Text Editing

- Text fields edit using a “field editor”
- You can customize text editing
  - Implement text editing delegate methods
  - Modify field editor’s attributes
  - Provide your own field editor
  - Attach formatters to your cell



Delegation/Notification

- Controls forward text editing messages

```
-(NSMenu *)control:(NSControl *)control
                 textShouldBeginEditing:(NSText *)text;
```

NSControlTextDidChangeNotification

- Each control defines its own unique notification and delegate messages

Drawing and Events

- Draw by overriding

```
-(void)drawWithFrame:(NSRect)cellFrame
                      inView:(NSView *)controlView;
```

- Handle mouse events

```
-(BOOL)trackMouse:(NSEvent *)theEvent
                   inRect:(NSRect)cellFrame
                   ofView:(NSView *)controlView
             untilMouseUp:(BOOL)flag;
```

- Determining size and location

```
-(NSSize)cellSizeForBounds:(NSRect)bounds;
```

Context Menus

- Control uses the cells menu in favor of its own
- Each can provide a static default menu

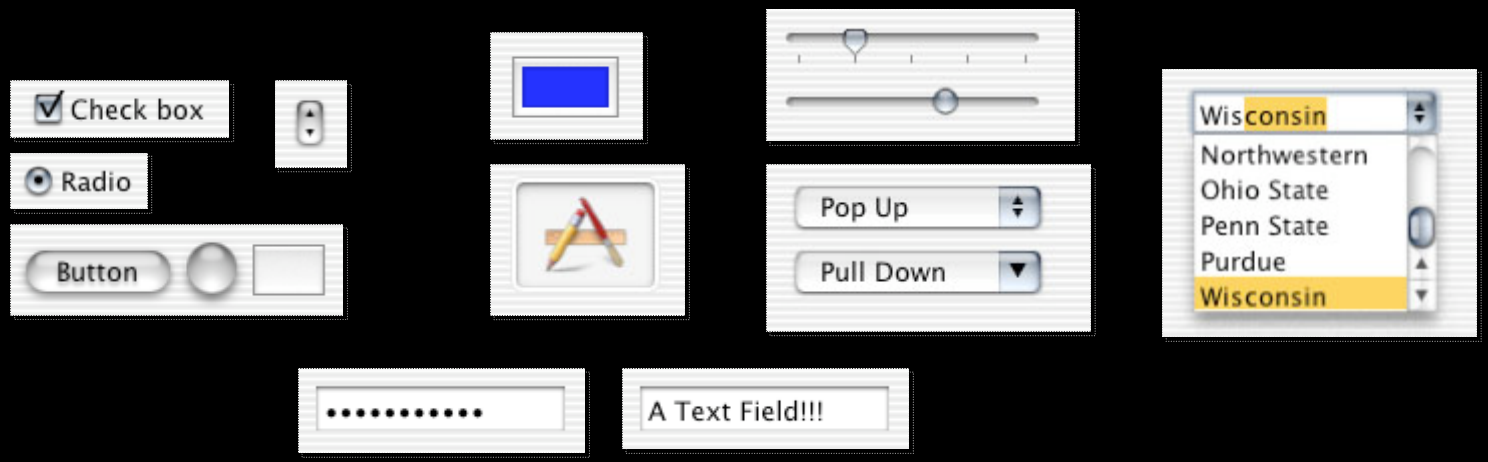
```
-(void)setMenu:(NSMenu *)menu;
-(NSMenu *)menu;
```

- . . or provide one dynamically

```
-(NSMenu *)menuForEvent:(NSEvent *)theEvent;

-(NSMenu *)menuForEvent:(NSEvent *)theEvent
                 inRect:(NSRect)cellFrame
                 ofView:(NSView *)controlView;
```

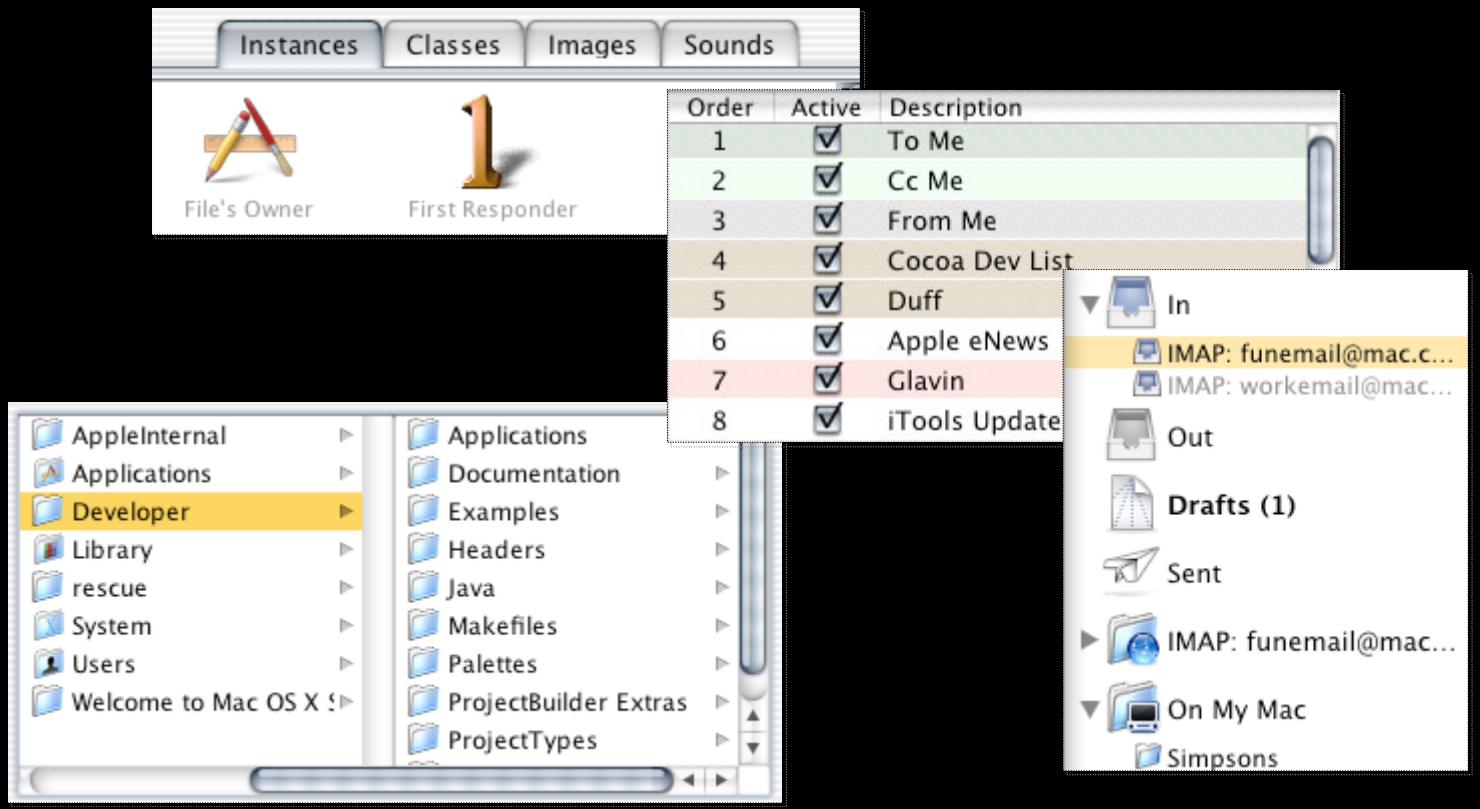
Simple Controls

- Cocoa has many simple controls
  - Easy to use
  - Do not require subclassing



Demo

More Controls



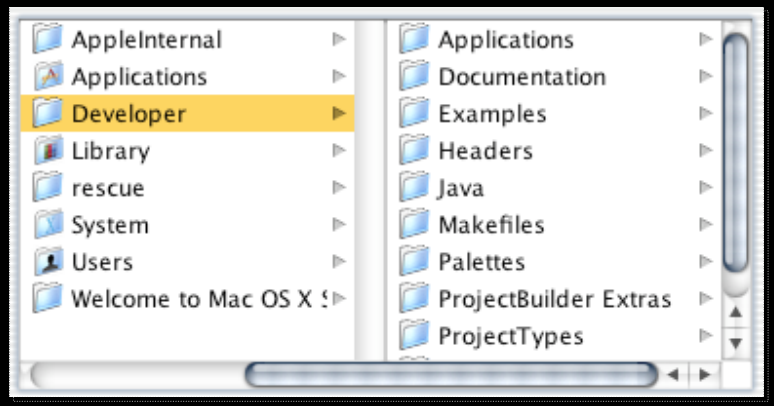
NSTabView

- A “view swapper”
- Manages NSTabViewItems
  - Label
  - View
- Support multiple styles
  - Tabs on: Top, bottom, left, right
  - Regular or small control size

NSMatrix

- Manages a group of NSCells
- Matrix has its own action and double action
- Usually created in IB (opt + drag)
- Can be created programmatically
    - Provide each cell, or a prototype
- Lots of API
    - Manage rows, columns, cells
    - Arrange cells, customize appearance

NSBrowser



- Displays hierarchical data
- Each column is an NSMatrix
- Matrices are filled with NSBrowserCells
- Data provided by a delegate
- Lots of API
  - Setting and getting the selection
  - Accessing cells, and matrices
  - Customize appearance

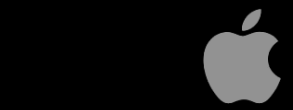
Loading Data

- Active (immediately)

```
-(void)browser:(NSBrowser *)browser
        createRowsForColumn:(int)numberOfRows
                   inMatrix:(NSMatrix *)matrix;
```

- Passive (lazy)

```
-(int)browser:(NSBrowser *)browser
    numberOfRowsInColumn:(int)columnIndex
                inMatrix:(NSMatrix *)matrix;

-(void)browser:(NSBrowser *)browser
          willDisplayCell:(id)cell
                    atRow:(int)row column:(int)col;
```



NSBrowserCell

- Data Properties

```
-(void)setLoaded:(BOOL)loaded;
-(BOOL)isLoaded;

-(void)setLeaf:(BOOL)leaf;
-(BOOL)isLeaf;
```

NSTableView

- Depends heavily on a data source
- Primarily row and column oriented
- Highly customizable
- Replaceable parts
  - NSTableColumn
  - NSTableHeaderView
  - NSTableHeaderCell

NSTableColumn

- Recognized by Identifier
- Provides data cell

```
-(void)setDataCell:(NSCell *)dataCell;
-(NSCell *)dataCellForRow:(int)row;
        // default implementation returns a shared cell
```

- Provides header cell

Data Source

- Required Methods

```
-(int)numberOfRowsInTableView:(NSTableView *)aTableView;

-(id)tableView:(NSTableView *)aTableView
        objectValueForTableColumn:(NSTableColumn *)column
                              row:(int)row;
```

- When data changes . . .

NSTableView:

```
-(void)reloadData;

-(void)noteNumberOfRowsChanged;
```

Object Value

- Sometimes API needs a generic type
- Every NSCell implements

```
-(void)setObjectValue:(id)objectValue;
-(id)objectValue;
```

- Example direct usage

```
[sliderCell setObjectValue:
        [NSNumber numberWithFloat:3.14]];

[textCell setObjectValue:
        [NSString stringWithFormat:@"Homer"]];
```

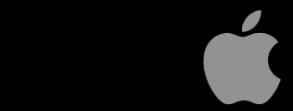
Data Source

- Optional Editing Method

```
-(id)tableView:(NSTableView *)aTableView
    setObjectValue:(id)newObjectValue
    forTableColumn:(NSTableColumn *)column
               row:(int)row;
```

- Optional Drag and Drop

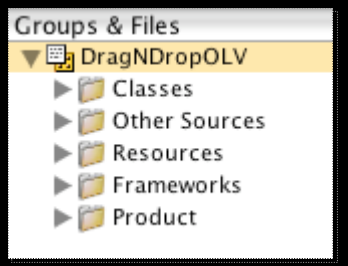
```
-(BOOL)tableView:(NSTableView *)aTableView
    writeRows:(NSArray *)rowNumbers
    toPasteboard:(NSTableColumn *)column

-(BOOL)tableView:(NSTableView *)aTableView
    acceptDrop:(id <NSDraggingInfo>)info
           row:(int)row
    dropOperation:(NSTableViewDropOperation)op;
```



Delegate/Notifications

- Delegation
  - Will display cell
  - Should edit table column
  - Row/column selection control
  - Did click/drag table header
- Notifications
  - Selection will/did change
  - Column did move/resize

NSOutlineView



- Similar to NSTableView, except . . .
  - Items have expandable property
  - API Deals in ‘items’ instead of rows

```
-(void)reloadItem:(id)item;
-(int)rowForItem:(id)item;
```

- Some API is specific to NSOutlineView
  - Expanding and collapsing items
  - Persisting the expanded items

Data Source

- Required Methods

```
-(int)outlineView:(NSOutlineView *)anOutlineView
        numberOfChildrenOfItem:(id)item;

-(id)outlineView:(NSOutlineView *)anOutlineView
    objectValueForTableColumn:(NSTableColumn *)tableCol
                       byItem:(id)item;

-(id)outlineView:(NSOutlineView *)anOutlineView
                      child:(int)childIndex
                     ofItem:(id)item;

-(BOOL)outlineView:(NSOutlineView *)anOutlineView
            isItemExpandable:(int)childIndex;

// nil represents the root item
```

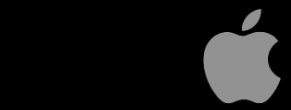
Demo

Cocoa Accessibility

- Keyboard accessibility
- Accessibility APIs

Keyboard Accessibility

- Setting up a keyboard loop
- Getting/displaying keyboard focus
- Handling keyboard events



Getting Keyboard Focus

- Tell the system you want focus

```
-(BOOL)acceptsFirstResponder;
```

- . . and when you want it

```
-(BOOL)needsPanelToBecomeKey;
        // Return YES to behave more like text fields
        // Return NO to behave more like a button
```

Display Keyboard Focus

- Knowing you have focus

```
-(BOOL)becomeFirstResponder;
-(BOOL)resignFirstResponder;

isFirstResp = ([[self window] firstResponder]==self);
```

- Drawing a focus ring

```
if ((isFirstResp && [[self window] isKeyWindow])) {
    [NSGraphicsContext saveGraphicsState];
    NSSetFocusRingStyle(…);
    …
    [NSGraphicsContext restoreGraphicsContext];
}
```

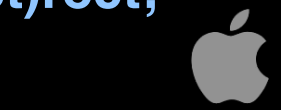
Display Keyboard Focus

- Update the focus ring when . . .
  - Your view becomes/resigns first responder
  - Your window changes key state

```
- (void)viewDidMoveToWindow {
    // Register for notifications…
    // NSWindowDidBecomeKeyNotification
    // NSWindowDidResignKeyNotification
}
```

- Invalidating the focus ring

```
-(void)setKeyboardFocusRingNeedsDisplayInRect:(NSRect)rect;
```



Keyboard Events

- Individual key strokes

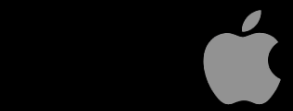
```
-(void)keyDown:(NSEvent *)theEvent;
```

- Specific methods

```
-(void)moveRight:(id)sender;      // arrow key
-(void)performClick:(id)sender;   // space bar
-(void)cancel:(id)sender;         // escape key
```

Keyboard Loops

- Set up the keyboard loop yourself
  - Set NSWindow's initialFirstResponder
  - Set each view's nextKeyView
- Let Cocoa compute the keyboard loop
  - If initialFirstResponder is unset
- Don't forget . . .
  - Update loop when adding/removing views
  - Test "All controls" mode



Demo

Accessorizing Cocoa Applications

Mike Engber
Cocoa Engineer

Accessibility—Overview

- Accessibility APIs
  - Examining other applications
- Clients
  - Assistive applications
  - Details covered in Session 009
- Targets
  - Cocoa and Carbon applications

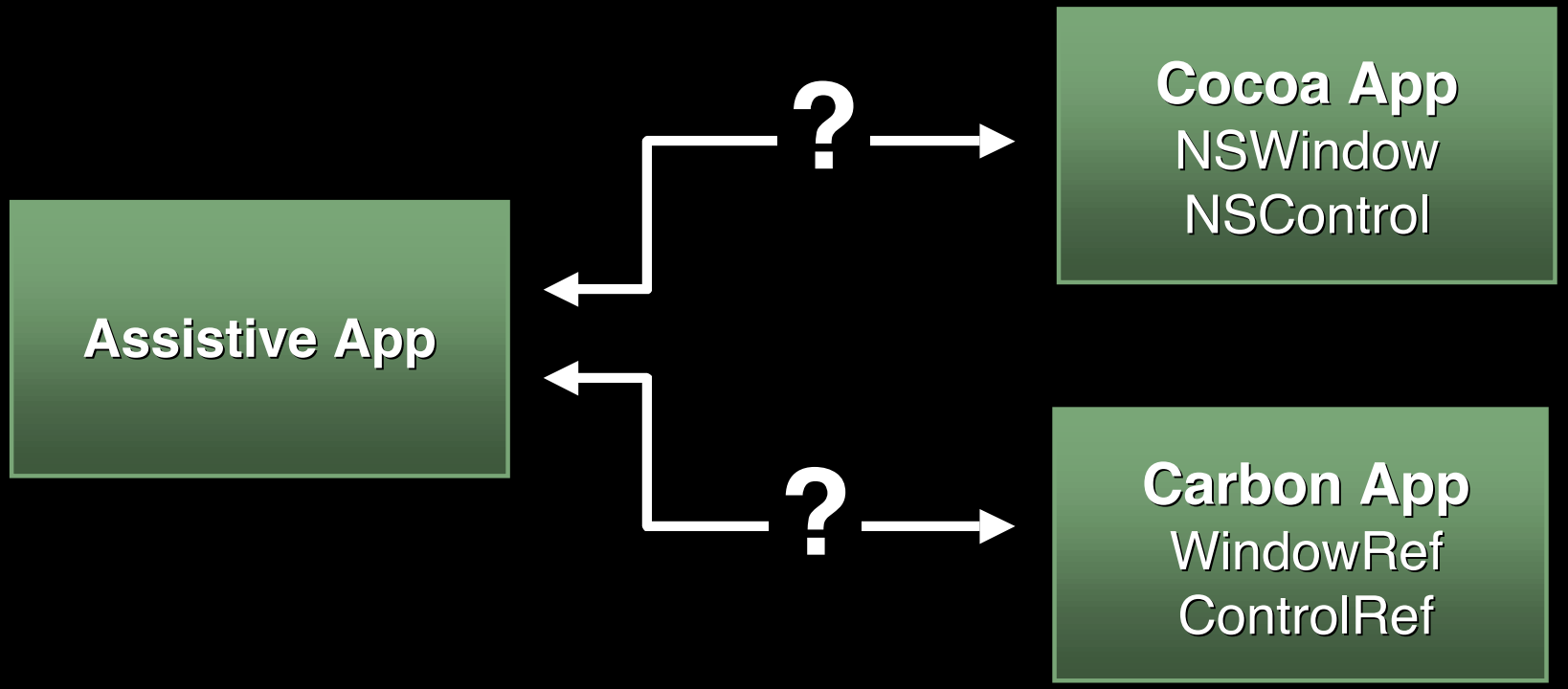
Demo

Kevin Aitken
Speech Engineer

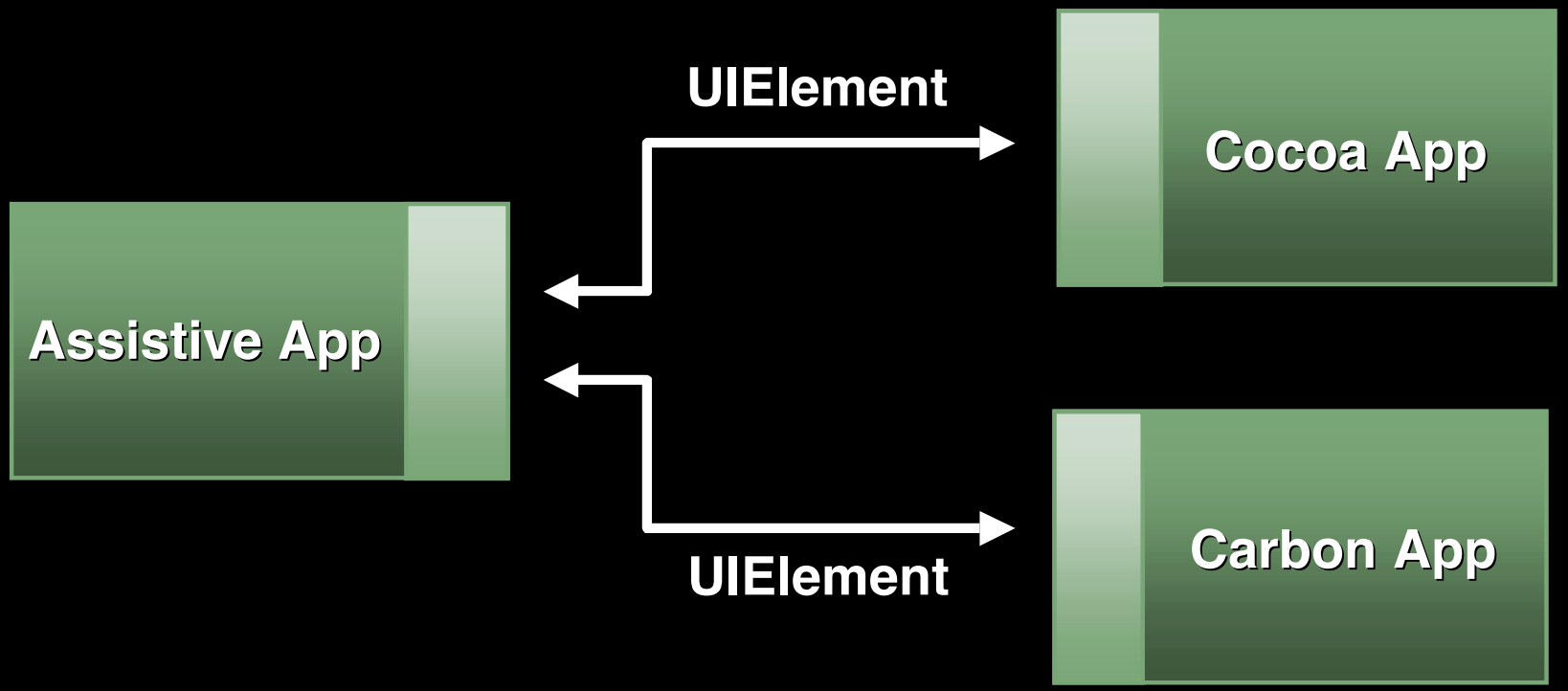
Accessibility

- Standard user interface elements
  - Already accessible
- Custom elements
  - Most will be accessible
- Unusual custom elements
  - NSAccessibility Protocol
  - Carbon equivalent is covered in Session 209

UI Representation—Problem

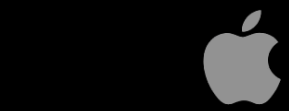


UI Representation—Solution



UIElements

- Hierarchy of UIElements
- Attributes
  - **NSAccessibilityTitleAttribute**
  - **NSAccessibilityValueAttribute**
  - **NSAccessibilityChildrenAttribute**
- Actions
  - **NSAccessibilityPressAction**
  - **NSAccessibilityIncrementAction**



NSAccessibility Protocol

- Informal protocol on NSObject
- Implemented by standard interface elements
  - NSWindow
  - NSView
  - NSControl
  - NSCell

Attribute Methods—Part 1

```
-(NSArray *) accessibilityAttributeNames

-(id) accessibilityAttributeValue:(NSString *)attr

(BOOL) accessibilityIsAttributeSettable:(NSString *)attr

-(void) accessibilitySetValue: (id)value
                 forAttribute: (NSString *)attr
```

Attribute Methods—Part 2

- Most likely to override attribute value
- E.g., title for a custom view

```
If ([attr isEqualToString:NSAccessibilityTitleAttribute]){
        return _myTitle;
} else {
        [super accessibilityAttributeValue:attr];
}
```

- Avoid new kinds of attributes

Action Methods—Part 1

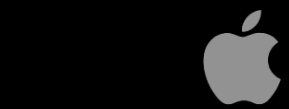
- (NSArray *) accessibilityActionNames

- (NSString *) accessibilityActionDescription:(NSString *)attr

- (void) accessibilityPerformAction: (NSString *)attr

Action Methods—Part 2

- Rarely overridden
- Action ≈ mouse click
    - Very simple, no arguments
- Setting attributes often suffices
    - No select text action
    - Instead, set selected text range



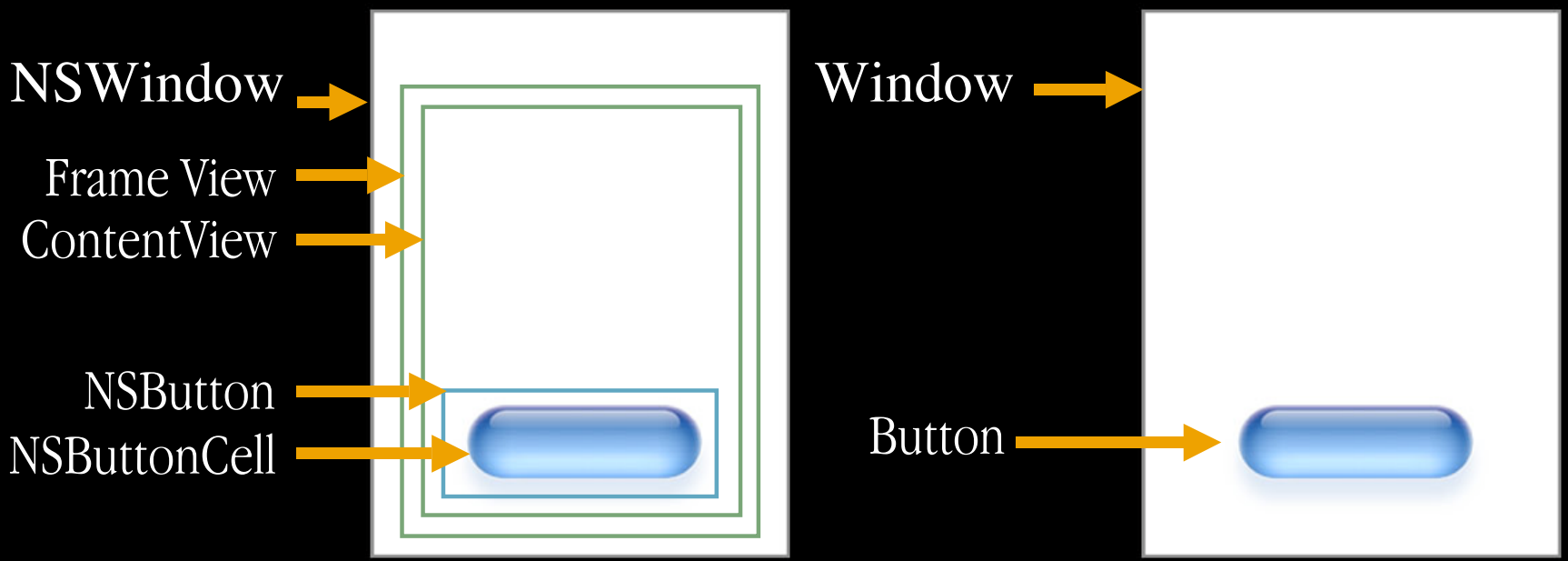
Demo

Kevin Aitken
Speech Engineer

Ignored UIElements—Part 1

Actual Hierarchy

Accessibility Hierarchy



Ignored UIElements—Part 2

```
-(BOOL)accessibilityIsIgnored

id NSAccessibilityUnignoredAncestor (id element)

NSArray *NSAccessibilityUnignoredChildren
                                (NSArray *originalChildren)
```

Hit/Focus Testing

- Accessibility APIs provide
  - UIElement under a point
  - UIElement with keyboard focus
- Views detected automatically
- UIElements handle their sub parts

Hit/Focus Testing

- (id) **accessibilityHitTest:**(NSPoint)point
- (id) **accessibilityFocusTest**

- Only called if you contain the point/focus
- Return child or self

Notifications—Part 1

- Application events

  NSAccessibilityApplicationActivatedNotification

- Window events

  NSAccessibilityWindowMiniaturizedNotification
  NSAccessibilityFocusedUIElementChangedNotification
  NSAccessibilityUIElementDestroyedNotification

Notifications—Part 2

```
void NSAccessibilityPostNotification
                (id element, NSString *notification)

NSAccessibilityValueChangedNotification
```

Role Attribute

- NSAccessibilityRoleAttribute
  - Describes the kind of UIElement
  - Elements with the same role should have the same attributes and actions
  - Avoid inventing new roles
- Instance specific information—Title and help

Summary

- Standard user interface elements
  - Already accessible
- Custom elements
  - Most will be accessible
- Unusual custom elements
  - NSAccessibility Protocol

Documentation

Controls

- Controls and Cells
  - Buttons
  - Browsers
  - Tables
  - Etc . . .

Documentation > Cocoa > User Interface Elements > Controls and Cells
developer.apple.com/techpubs/macosx/Cocoa/TasksAndConcepts/ProgrammingTopics/ControlCell/index.html

Documentation

Accessibility

- Making Your Application Accessible to Users With Disabilities

ADC Member Site > Download Software > “Jaguar” Mac OS X > Docs
connect.apple.com

- Accessibility Reference for Assistive Applications

“Jaguar” Mac OS X Developer Tools CD
/Developer/Documentation/ReleaseNotes/AssistiveAPI.html

For More Information

- O'Reilly "Learning Cocoa" and "Building Cocoa Applications: A Step-by-Step Guide"
- Cocoa Developer Documentation

  http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html
- Apple Customer Training

  http://train.apple.com/

Roadmap

300 Introduction to Cocoa: Overview and hands-on demo of Cocoa APIs	Room A1 **Mon., 5:00pm**
301 Cocoa: What's New: New features and API since 10.1	Civic **Tues., 9:00am**
302 Cocoa API Techniques: Understanding, leveraging, and extending	Hall 2 **Thurs., 9:00am**
305 Cocoa Drawing: 2D graphics in Cocoa: Images, bezier paths,..	Hall 2 **Fri., 10:30am**

Roadmap

013 Speech Technologies in Mac OS X: Deliver the speech experience in your app	Room A2 **Fri., 5:00pm**
FF016 Cocoa: Comments and suggestions for Cocoa	Room A1 **Fri., 5:00pm**

Who to Contact

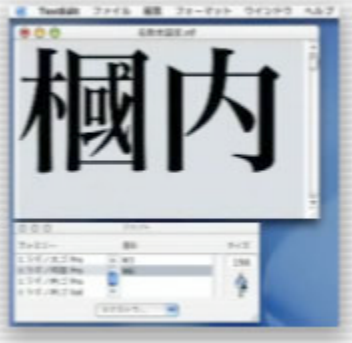
Heather Hickman
Cocoa Evangelist
hhickman@apple.com

Cocoa Feedback
For comments on Cocoa (not a discussion list)
cocoa-feedback@group.apple.com

Cocoa Discussion
For Cocoa-related discussion with developers worldwide
cocoa-dev@lists.apple.com

http://developer.apple.com/wwdc2002/urls.html

Q&A

Heather Hickman
Cocoa Evangelist
hhickman@apple.com

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002